



LearnDelphi.org

Coding Boot Camp

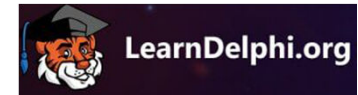
August 2022

Core 13

Time to work with Date and Time



Coding Boot Camp 2022

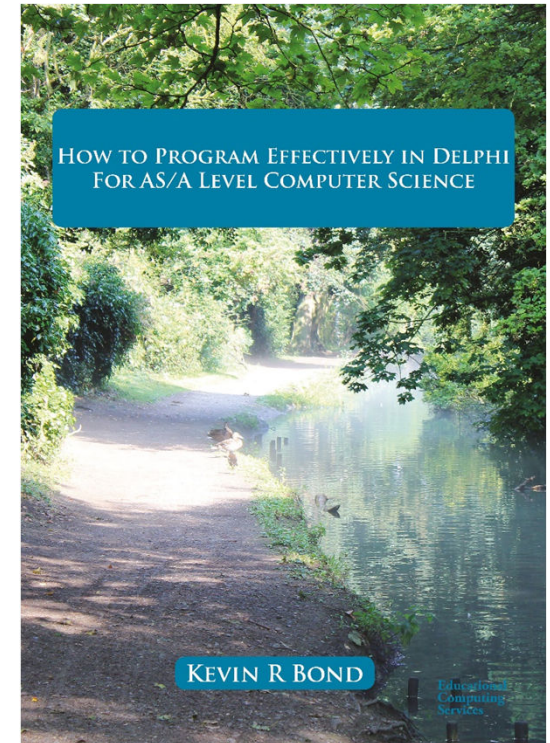


How to Program Effectively in Delphi for AS/A Level Computer Science

By Kevin R Bond

Available in both print and pdf formats from
<https://www.educational-computing.net>

Main site for information
<http://www.educational-computing.co.uk>



Length =1200 pages.
Suitable for all levels from
beginner to advanced developer

What is time – what this talk is not about?

❑ The ARROW Of TIME

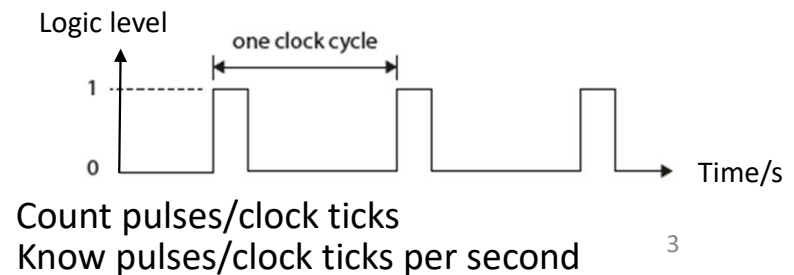
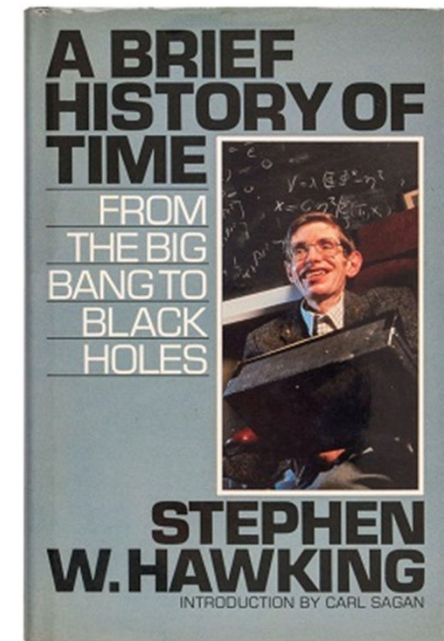
- The existence of a thermodynamic arrow of time implies that the system is highly ordered in one time direction only, which would by definition be the "past" – you can't unscramble eggs

❑ Entropy increases **when a substance is broken up into multiple parts**

❑ As one goes "forward" in time, 2nd Law of Thermodynamics says:

- the entropy of an isolated system can increase, but not decrease.

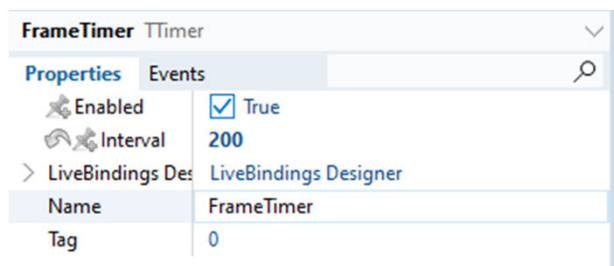
❑ Entropy measurement is a way of distinguishing the past from the future



Overview

- ❑ How to use a timing component, TTimer, from the Vcl.ExtCtrls unit and the FMX.Types unit, to create an event that can be responded to
- ❑ How to obtain a measurement of a time interval using
 - GetTickCount and GetTickCount64 from the WinAPI.Windows unit
 - TStopwatch, a high resolution timer, from the System.Diagnostics unit
- ❑ How to obtain the current date and time using the TDateTime data type from the System Unit
- ❑ Working with dates and times in Delphi using date and time routines found in the System. System.SysUtils and System.DateUtils units

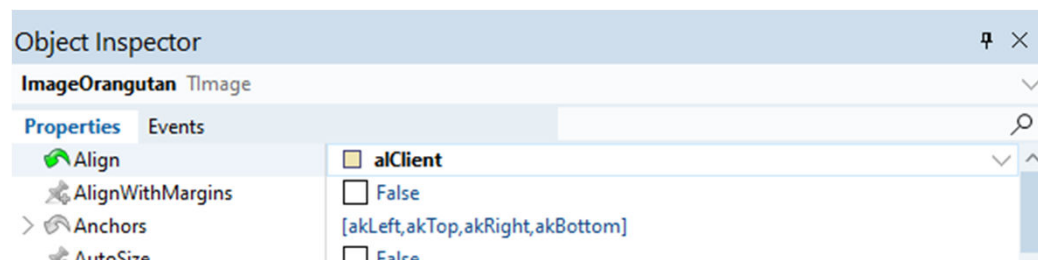
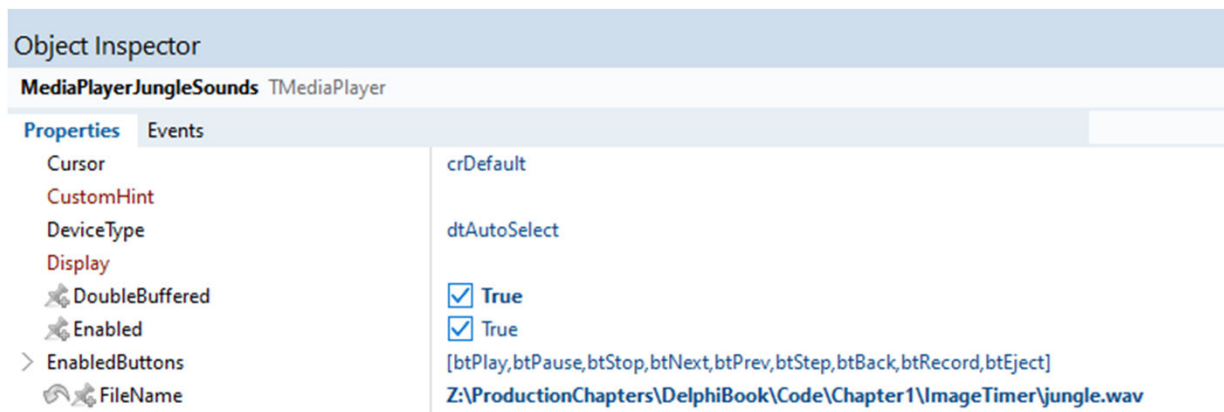
Using TTimer from Vcl.ExtCtrls unit



The standard VCL TTimer component is a wrapper for the Windows API timer functions SetTimer and KillTimer. TTimer simplifies the processing of the WM_TIMER messages by converting them into OnTimer events. A timer based on WM_TIMER message processing cannot provide a resolution better than 10 milliseconds.

```
Procedure TMainForm.FrameTimerTimer(Sender: TObject);
Begin
  MediaPlayerJungleSounds.Play;
  Counter := (Counter + 1) Mod 42;
  NumberString := '00' + IntToStr(Counter + 16);
  FileName := 'IMGP' + NumberString;
  MainForm.Caption := FileName;
  ImageOrangutan.Stretch := True;
  ImageOrangutan.Picture.LoadFromFile('Z:\ProductionChapters\DelphiBook\Code\Chapter1\Images\' + FileName + '.jpg');
End;
```

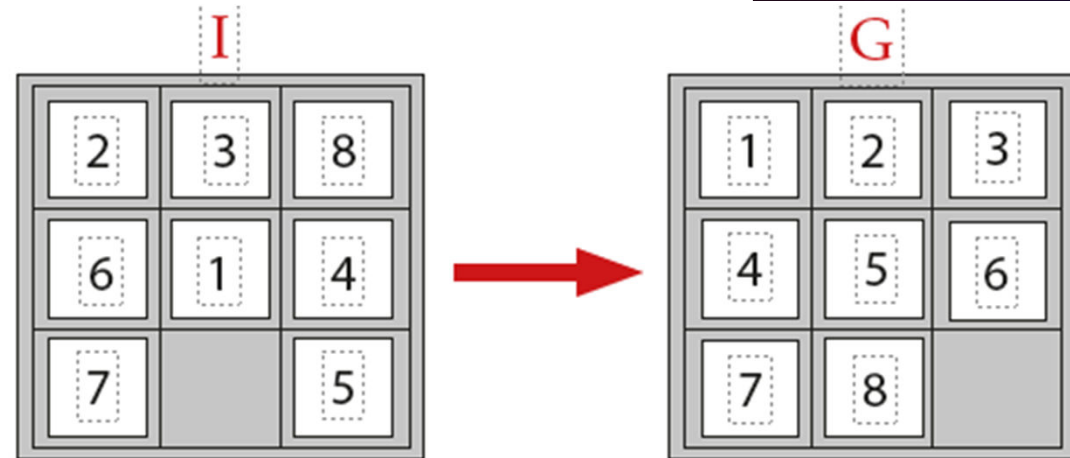
Using TTimer from Vcl.ExtCtrls unit



Computability and complexity

Non-computable problem

- ❑ Eight-puzzle problem.
- ❑ The puzzle has an initial state I.
- ❑ The objective is to reach the goal state G from the given initial state I by sliding the numbered tiles where this is possible.
- ❑ Unfortunately, there is no sequence of steps that will achieve this.
- ❑ Therefore, any algorithm that attempts to solve this problem will not terminate for the given initial and goal states.
- ❑ The problem is not solvable, and therefore the problem is non-computable.



The number of transpositions/exchanges required to turn 23861475 into 12345678 is 11, an odd number.

| | |
|----------|------------------|
| 23861475 | |
| 23614758 | 5 transpositions |
| 23614578 | 1 transposition |
| 23145678 | 3 transpositions |
| 21345678 | 1 transposition |
| 12345678 | 1 transposition |

The number of transpositions required to turn 12345678 into 12345678 is 0, an even number.

Odd to even is not possible!
Odd means that after any number of exchanges there will always be at least one tile out of sequence.

Computable solution

Sliding Puzzle 8

Tractable Sliding Puzzle 8
Educational Computing Services Ltd

| | | |
|---|---|---|
| 8 | 3 | 4 |
| 6 | 5 | 1 |
| 2 | 7 | |

Click or tap a tile adjacent by row or column to blank tile to swap their positions. The goal is to arrange the numbers on tiles sequentially from the top by row.

Sliding Puzzle 8

Tractable Sliding Puzzle 8
Educational Computing Services Ltd

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | |

Click or tap a tile adjacent by row or column to blank tile to swap their positions. The goal is to arrange the numbers on tiles sequentially from the top by row.

The number of transpositions required to turn 12345678 into 12345678 is 0, an even number.

The number of transpositions required to turn 83465127 into 12345678 is 16, an even number.

| | |
|----------|------------------|
| 83465127 | |
| 34651278 | 7 transpositions |
| 34512678 | 3 transpositions |
| 34125678 | 2 transpositions |
| 31245678 | 2 transpositions |
| 12345678 | 2 transpositions |

Computability and complexity

Complexity

Even if the problem is an algorithmic problem and it is computable, whether a solution can be produced may still depend on the **size of the input** and the **amount of work that the computer may need to do**.

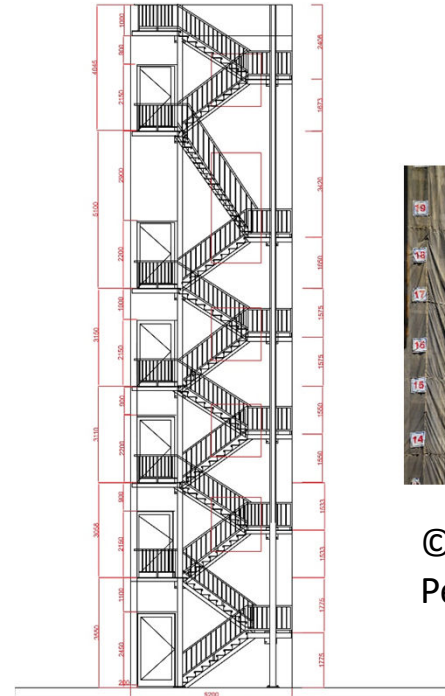
Time efficiency of an algorithm:

The time efficiency of an algorithm is a theoretical time measured by hand in computational steps arrived at by counting the number of steps the algorithm makes to complete a task.

More steps means the algorithm will take longer than another algorithm requiring fewer steps to complete the same task.



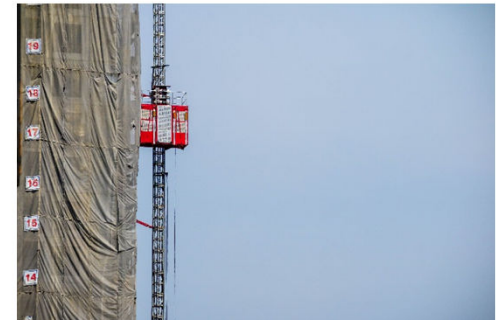
© Shutterstock / 145794110.
Permission obtained.



© Shutterstock / 349238483. Permission obtained.

**More steps means more
time consumed**

TotalTime = NoOfSteps x StepTime



© Shutterstock / 63152726
Permission obtained.

The table shows in pseudo-code two algorithms, algorithm 1 and algorithm 2, for calculating the sum of the first n natural numbers, i.e.

$$1 + 2 + 3 + 4 + \dots + (n-1) + n$$

The first algorithm involves a number of steps which varies with n as follows

1. 3 assignment steps (\leftarrow)
2. An input step (USERINPUT)
3. An output step (OUTPUT)
4. A comparison step ($\text{Count} < n$)
5. 2 assignment steps (\leftarrow) repeated n times (WHILE loop)
6. 2 addition steps ($+$) repeated n times (WHILE loop)

| Algorithm 1 | Algorithm 2 |
|---|---|
| <pre> 1 n ← USERINPUT Sum ← 0 Count ← 0 WHILE Count < n Count ← Count + 1 Sum ← Sum + Count ENDWHILE OUTPUT Sum </pre> | <pre> 1 2 3 n ← USERINPUT Sum ← (n * (n + 1)) / 2 OUTPUT Sum </pre> |

The second algorithm involves the following 7 steps whatever the value of n :

1. 2 assignment steps (\leftarrow)
2. An input step (USERINPUT)
3. An output step (OUTPUT Sum)
4. An addition step ($+$)
5. A multiplication step ($*$)
6. A division step ($/2$)

Setting aside whether or not each of the different type of steps (assignment, addition, etc) are “worth” different amounts of time, algorithm 1 takes more steps than algorithm 2 to do the same task.

```

Z:\ProductionChapters\DelphiBook\Del...
Input n: 10000000
Time (in milliseconds): 25
Time (in milliseconds): 0
Another go (Y/N)? :
Input n: 100000000
Time (in milliseconds): 254
Time (in milliseconds): 0
Another go (Y/N)? :
Input n: 1000000000
Time (in milliseconds): 2548
Time (in milliseconds): 0
Another go (Y/N)? :

```

```
Program SumToNLoopingAlgorithm1;
{$APPTYPE CONSOLE}
{$R *.res}
Uses
  System.SysUtils, System.Diagnostics;
Var
  Stopwatch : TStopWatch;
  n, Count, Sum : Int64;
  Ch : Char;
Begin
  Stopwatch := TStopWatch.Create;
  Repeat
    Write('Input n: ');
    Readln(n);
    Stopwatch.Start;
    Count := 0;
    Sum := 0;
    While Count < n
      Do
        Begin
          Count := Count + 1;
          Sum := Sum + Count;
        End;
    Stopwatch.Stop;
    Writeln(Format('Time (in milliseconds): %d ', [StopWatch.ElapsedMilliseconds]));
    Stopwatch.Reset;
    Stopwatch.Start;
    Sum := n*(n + 1) Div 2;
    Stopwatch.Stop;
    Writeln(Format('Time (in milliseconds): %d ', [StopWatch.ElapsedMilliseconds]));
    Write('Another go (Y/N)? ');
    Readln(Ch);
  Until Ch In ['n', 'N'];
End.
```

TStopWatch provides a high resolution timer

The accuracy of high-resolution timers is around a few hundred nanoseconds. A nanosecond is a unit of time representing 0.000000001 seconds -- or 1 billionth of a second.

GetTickCount and GetTickCount64


Function GetTickCount and function GetTickCount64 return the number of milliseconds since the operating system started.

GetTickCount returns a 32 bit DWord which wraps every 49.7 days.

Therefore, it is better to use GetTickCount64 (available since Windows Vista) which uses 64 bits.

Unfortunately, the resolution of the system timer which GetTickCount and GetTickCount64 rely upon is typically in the range of 10 milliseconds to 16 milliseconds.

```
Program GetTickCountProject;
{$APPTYPE CONSOLE}
{$R *.res}
Uses
  System.SysUtils,
  WinAPI.Windows;
Var
  Ticks :UInt64;
Begin
  Ticks := GetTickCount;
  Sleep(1000);
  {Program execution paused for 1000 milliseconds}
  Writeln('ProcessingTime: ',
    (GetTickCount - Ticks).ToString, ' milliseconds');
  Readln;
End.
```

 Z:\ProductionChapters\DelphiBook\DelphiBoc
ProcessingTime: 1000 milliseconds

```
Program GetTickCount64Project;
{$APPTYPE CONSOLE}
{$R *.res}
uses
  System.SysUtils,
  WinAPI.Windows;
Var
  Ticks :UInt64;
Begin
  Ticks := GetTickCount64;
  Sleep(1000);
  {Program execution paused for 1000 milliseconds}
  Writeln('ProcessingTime: ',
    (GetTickCount64 - Ticks).ToString, ' milliseconds');
  Readln;
End.
```

Try Sleep(10); Sleep(15);

Time and Dates

Values may be dates, e.g. 28/05/2016, or a time of day, e.g. 15:35:16.

Internally, these are just bit patterns.

For these bit patterns to have the meaning date or the meaning time of day, the bit patterns must be data typed.

This requires a data type date and a data type time or a combination of the two, a data type datetime.

Delphi uses the latter and names this data type **TDateTime**.

| Data type | Description |
|------------------|---|
| TDateTime | Data type double, with the date as the integral part, and time as fractional part. The date is stored as the number of days since 30 December 1899. |
| TDate | Represents a special type of TDateTime value that has no decimal part. A TDate value represents the number of days that have elapsed since 30/12/1899. |
| TTime | Represents a special type of TDateTime that has only a fractional part (and no integral part). A TTime value represents the fraction of a 24-hour day. For example, 0.25 represents 06:00 hours, 0.5 represents 12:00 hours, and 0.75 represents 18:00 hours. |

TDateTime is defined in the System unit as **Type = Type Double;**

This form of type definition forces the compiler to create a new distinct type called **TDateTime**.

Time and Dates

| | |
|----------------------|--|
| Now | Returns the current date and time as a value of type Double , e.g. 43707.45404 to 5 decimal places |
| Date | Returns only the date as a value of type Double , e.g. 43707.00000 to 5 decimal places |
| Time | Returns only the time as a value of type Double , e.g. 0.45404 to 5 decimal places |
| DateTimeToStr | Converts a date and time value into a string e.g. 30/08/2019 10:53:48 |
| DateToStr | Converts a date value into a string e.g. 30/08/2019 |
| TimeToStr | Converts a time value into a string e.g. 10:53:48 |

The above routines are functions.

```

Program DateTimeRoutines;
{$APPTYPE CONSOLE}
Uses
    SysUtils;
Begin
    Writeln(Now : 10 : 5);
    Writeln(Date : 10 : 5);
    Writeln(Time : 7 : 5);
    Writeln(DateTimeToStr(Now));
    Writeln(DateToStr(Date));
    Writeln(TimeToStr(Time));
    Readln;
End.

```

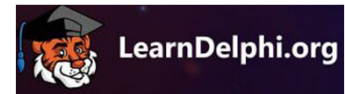
Z:\ProductionChapters\De

```

44791.33938
44791.00000
0.33938
18/08/2022 08:08:42
18/08/2022
08:08:42

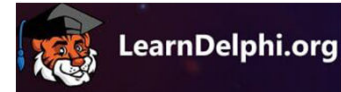
```

Time and Dates



| Specifier | Displays |
|-----------|--|
| c | Displays the date using the format given by the ShortDateFormat global variable, followed by the time using the format given by the LongTimeFormat global variable. The time is not displayed if the date-time value indicates midnight precisely. |
| d | Displays the day as a number without a leading zero (1-31). |
| dd | Displays the day as a number with a leading zero (01-31). |
| ddd | Displays the day as an abbreviation (Sun-Sat) using the strings given by the ShortDayNames global variable. |
| dddd | Displays the day as a full name (Sunday-Saturday) using the strings given by the LongDayNames global variable. |
| dddddd | Displays the date using the format given by the ShortDateFormat global variable. |
| m | Displays the month as a number without a leading zero (1-12). If the m specifier immediately follows an h or hh specifier, the minute rather than the month is displayed. |
| mm | Displays the month as a number with a leading zero (01-12). If the mm specifier immediately follows an h or hh specifier, the minute rather than the month is displayed. |
| mmm | Displays the month as an abbreviation (Jan-Dec) using the strings given by the ShortMonthNames global variable. |
| mmm | Displays the month as a full name (January-December) using the strings given by the LongMonthNames global variable. |
| yy | Displays the year as a two-digit number (00-99). |
| yyyy | Displays the year as a four-digit number (0000-9999). |

Time and Dates



```
Program DateTimeWithFormatting;
```

```
{$APPTYPE CONSOLE}
```

```
{$R *.res}
```

```
Uses
```

```
    System.SysUtils;
```

```
Begin
```

```
    Writeln(FormatDateTime('dddd mmmm yyyy h:n:s:z ampm', Now , FormatSettings));
```

```
    FormatSettings.DateSeparator := ' ';
```

```
    FormatSettings.TimeSeparator := ':';
```

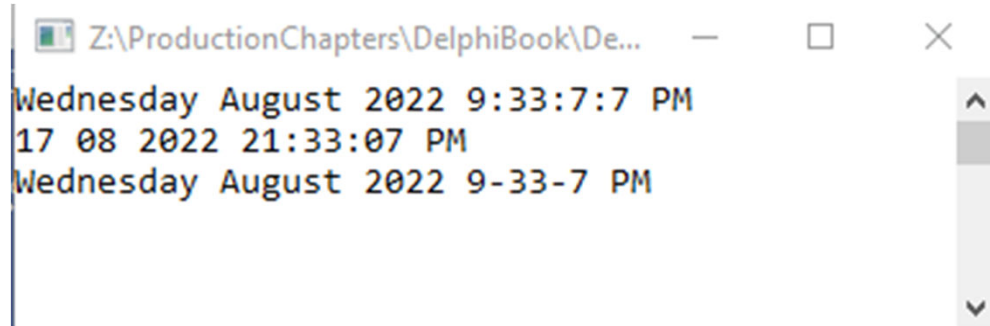
```
    Writeln(FormatDateTime('c ampm', Now , FormatSettings));
```

```
    FormatSettings.DateSeparator := ' '; FormatSettings.TimeSeparator := '-';
```

```
    Writeln(FormatDateTime('dddd/mmmm/yyyy h:n:s ampm', Now , FormatSettings));
```

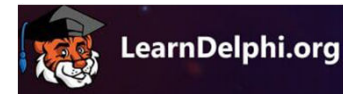
```
    Readln;
```

```
End.
```

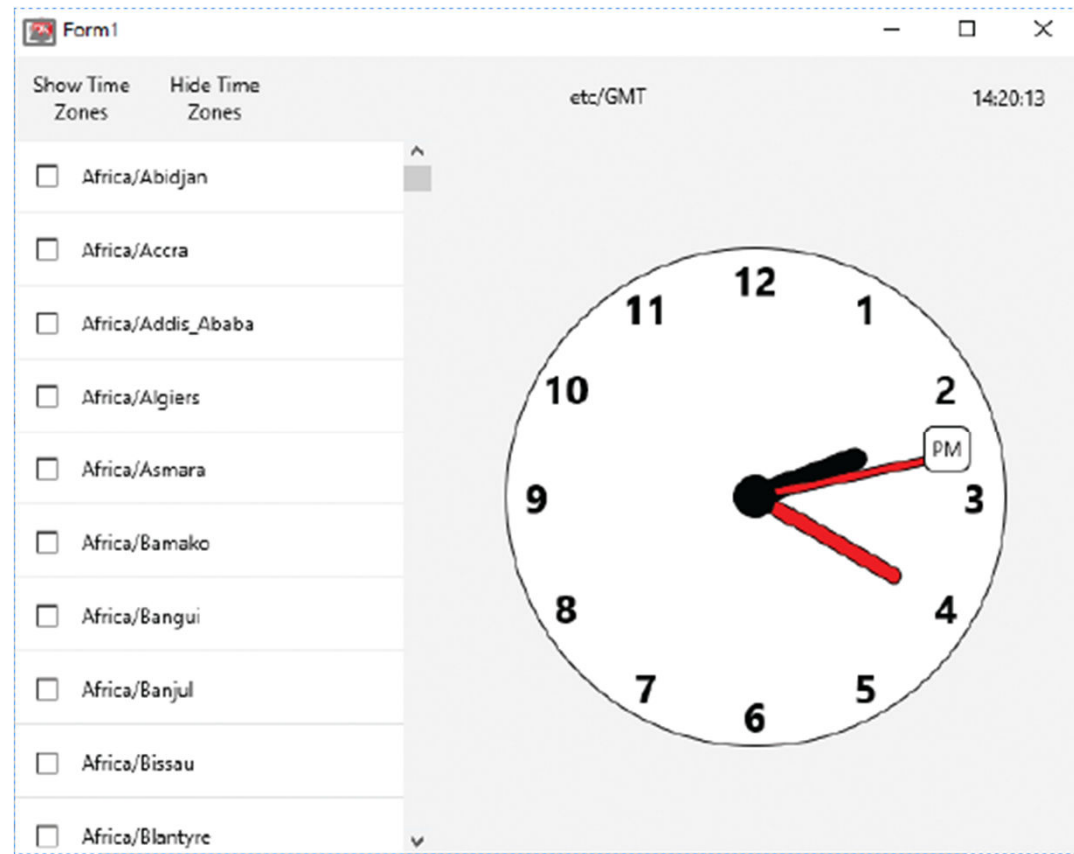


FormatSettings is a global variable of **TFormatSettings** record type in Delphi which is used to get local information such as DateTimeFormat, CurrencyFormat, DecimalSeparator etc. This variable was introduced in Delphi XE.

Analogue clock built using FireMonkey components

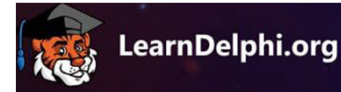


```
Procedure TForm1.Timer1Timer(Sender: TObject);
Var
  Hour, TwelveHour, Minute, Second : Word;
  strHour, strMinute, strSecond : String;
  DateTime : TDateTime;
Begin
  DateTime := Now;
  Hour := HourOf(DateTime);
  TwelveHour := Hour Mod 12;
  Minute := MinuteOf(DateTime);
  Second := SecondOf(DateTime);
  rrHour.RotationAngle := 30 * TwelveHour + Round(Minute/2.17);
  rrMinute.RotationAngle := 6 * Minute;
  rrSecond.RotationAngle := 6 * Second;
  strHour := Hour.ToString;
  strMinute := Minute.ToString;
  strSecond := Second.ToString;
  If (Length(strMinute) < 2)
    Then strMinute := '0' + strMinute;
  If (Length(strSecond) < 2)
    Then strSecond := '0' + strSecond;
  If (Length(strHour) < 2)
    Then strHour := '0' + strHour;
  Text3.Text := strHour + ':' + strMinute + ':' + strSecond;
End;
```



Based on an idea from Harry Stahl's book Cross-Platform Development with Delphi 10.2 & FireMonkey.
With Harry's permission extended this cross-platform application by adding time zones.

Download information for programs used in this presentation



Source code for these programs may be downloaded from a url supplied by contacting drbond@educational-computing.co.uk or from the LearnDelphi.org site. Look for the .zip extension of the following

